Typescript programs

Typescript looks like this :

var name = “Susan”,

age = 25,

hasCode = true;

We could write the following:

let name: string = "Susan",

age: number = 25,

hasCode: boolean = true;

1. var num:number = 12

console.log(num)

On compiling , it will generate following JavaScript code.

//Generated by typescript 1.8.10

var num = 12;

console.log(num);

1. var message:string = "Hello World"

console.log(message)

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var message = "Hello World";

console.log(message);

1. TypeScript and Object Orientation

class Greeting {

greet():void {

console.log("Hello World!!!")

}

}

var obj = new Greeting();

obj.greet();

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var Greeting = (function () {

function Greeting() {

}

Greeting.prototype.greet = function () {

console.log("Hello World!!!");

};

return Greeting;

}());

var obj = new Greeting();

obj.greet()

1. Variables

var name:string = "John";

var score1:number = 50;

var score2:number = 42.50

var sum = score1 + score2

console.log("name"+name)

console.log("first score: "+score1)

console.log("second score: "+score2)

console.log("sum of the scores: "+sum)

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var name = "John";

var score1 = 50;

var score2 = 42.50;

var sum = score1 + score2;

console.log("name" + name);

console.log("first score: " + score1);

console.log("second score : " + score2);

console.log("sum of the scores: " + sum);

1. Variable Scope

var global\_num = 12 //global variable

class Numbers {

num\_val = 13; //class variable

static sval = 10; //static field

storeNum():void {

var local\_num = 14; //local variable

}

}

console.log("Global num: "+global\_num)

console.log(Numbers.sval) //static variable

var obj = new Numbers();

console.log("Global num: "+obj.num\_val)

On transpiling, the following JavaScript code is generated −

var global\_num = 12; //global variable

var Numbers = (function () {

function Numbers() {

this.num\_val = 13; //class variable

}

Numbers.prototype.storeNum = function () {

var local\_num = 14; //local variable

};

Numbers.sval = 10; //static field

return Numbers;

}());

console.log("Global num: " + global\_num);

console.log(Numbers.sval); //static variable

var obj = new Numbers();

console.log("Global num: " + obj.num\_val);

Interfaces

==========

interface IPerson {

firstName:string,

lastName:string,

sayHi: ()=>string

}

var customer:IPerson = {

firstName:"Tom",

lastName:"Hanks",

sayHi: ():string =>{return "Hi there"}

}

console.log("Customer Object ")

console.log(customer.firstName)

console.log(customer.lastName)

console.log(customer.sayHi())

var employee:IPerson = {

firstName:"Jim",

lastName:"Blakes",

sayHi: ():string =>{return "Hello!!!"}

}

console.log("Employee Object ")

console.log(employee.firstName) console.log(employee.lastName)

//Generated by typescript 1.8.10

var customer = { firstName: "Tom", lastName: "Hanks",

sayHi: function () { return "Hi there"; }

};

console.log("Customer Object ");

console.log(customer.firstName);

console.log(customer.lastName);

console.log(customer.sayHi());

var employee = { firstName: "Jim", lastName: "Blakes",

sayHi: function () { return "Hello!!!"; } };

console.log("Employee Object ");

console.log(employee.firstName);

console.log(employee.lastName);

simple Interface Inheritance

interface Person {

age:number

}

interface Musician extends Person {

instrument:string

}

var drummer = <Musician>{};

drummer.age = 27

drummer.instrument = "Drums"

console.log("Age: "+drummer.age) console.log("Instrument: "+drummer.instrument)

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var drummer = {};

drummer.age = 27;

drummer.instrument = "Drums";

console.log("Age: " + drummer.age);

console.log("Instrument: " + drummer.instrument);

Multiple Inheritance

interface IParent1 {

v1:number

}

interface IParent2 {

v2:number

}

interface Child extends IParent1, IParent2 { }

var Iobj:Child = { v1:12, v2:23}

console.log("value 1: "+this.v1+" value 2: "+this.v2)

//Generated by typescript 1.8.10

var Iobj = { v1: 12, v2: 23 };

console.log("value 1: " + this.v1 + " value 2: " + this.v2);

Classes

class Car {

//field

engine:string;

//constructor

constructor(engine:string) {

this.engine = engine

}

//function

disp():void {

console.log("Function displays Engine is : "+this.engine)

}

}

//create an object

var obj = new Car("XXSY1")

//access the field

console.log("Reading attribute value Engine as : "+obj.engine)

//access the function

obj.disp()

/Generated by typescript 1.8.10

var Car = (function () {

//constructor

function Car(engine) {

this.engine = engine;

}

//function

Car.prototype.disp = function () {

console.log("Function displays Engine is : " + this.engine);

};

return Car;

}());

//create an object

var obj = new Car("XXSY1");

//access the field

console.log("Reading attribute value Engine as : " + obj.engine);

//access the function

obj.disp();

Class Inheritance

class Shape {

Area:number

constructor(a:number) {

this.Area = a

}

}

class Circle extends Shape {

disp():void {

console.log("Area of the circle: "+this.Area)

}

}

var obj = new Circle(223);

obj.disp()

On compiling, it will generate following JavaScript code.

//Generated by typescript 1.8.10

var \_\_extends = (this && this.\_\_extends) || function (d, b) {

for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];

function \_\_() { this.constructor = d; }

d.prototype = b === null ? Object.create(b) : (\_\_.prototype = b.prototype, new \_\_());

};

var Shape = (function () {

function Shape(a) {

this.Area = a;

}

return Shape;

}());

var Circle = (function (\_super) {

\_\_extends(Circle, \_super);

function Circle() {

\_super.apply(this, arguments);

}

Circle.prototype.disp = function () {

console.log("Area of the circle: " + this.Area);

};

return Circle;

}(Shape));

var obj = new Circle(223);

obj.disp();